Communication mechanisms for service-centric networking

Gasparyan, Mikael ; Schiller, Eryk ; Marandi, Ali ; Braun, Torsten

Abstract: L-SCN is a two-layered Service-Centric Networking (SCN) architecture. The L-SCN design splits the network into domains and specifies communication protocols for service provider information propagation. Nodes in a domain receive substantial knowledge about the available resources (e.g., CPU, RAM) and available services within the domain, while the communication between different domains is realized through supernodes. We extend L-SCN with new communication mechanisms, which improve the processing time and provide lower protocol overhead for service request processing. The two proposed mechanisms are named event-driven and provider-driven. The event-driven mechanism propagates service provider information based on an event (e.g., high overload). The provider-driven mechanism propagates service provider information periodically.

DOI: https://doi.org/10.1109/CCNC46108.2020.9045531

Communication Mechanisms for Service-Centric Networking

Mikael Gasparyan, Eryk Schiller, Ali Marandi, Torsten Braun
University of Bern, Switzerland
{gasparyan,schiller,marandi,braun}@inf.unibe.ch

***Abstract*—L-SCN is a two-layered Service-Centric Networking (SCN) architecture. The L-SCN design splits the network into domains and specifies communication protocols for service provider information propagation. Nodes in a domain receive substantial knowledge about the available resources (e.g., CPU, RAM) and available services within the domain, while the communication between different domains is realized through supernodes. We extend L-SCN with new communication mechanisms, which improve the processing time and provide lower protocol overhead for service request processing. The two proposed mechanisms are named event-driven and provider-driven. The event-driven mechanism propagates service provider information based on an event (e.g., high overload). The provider-driven mechanism propagates service provider information periodically.**

***Index Terms*—service-centric networking, icn, provider-driven, event-driven, service routing, future Internet**

I. Introduction

One of the most prominent future Internet architectures is Information-Centric Networking (ICN) [1], in which each piece of content is identified through a unique identifier. A content requester sends a request to fetch a given content using content identifiers, and requests are forwarded based on these content identifiers as well. Named Data Networking (NDN) is one of the most mature implementations of ICN.

The current service-oriented nature of the Internet pushed researchers to develop Service-Centric Networking (SCN) [2]. SCN keeps all the design primitives and architectural methods of ICN, but extends ICN with service support. SCN enables content provider entities in the network to also offer services that can be requested by service consumers. Generally, a service is a software function that requires input parameters and has an output result. In our context, a service requester sends a service request, which is forwarded to the corresponding service provider using NDN primitives. Finally, the result of the service output is sent to the service requester.

Previously, we have presented and evaluated an SCN two-layered routing architecture named L-SCN [3]. L-SCN divides a network into domains. Nodes in the same domain (i.e., intra-domain) possess a considerable amount of knowledge about services and resources available in their domain. Domains are connected to each other through supernodes. A supernode is a member of a domain and is connected to supernodes of other neighboring domains. Supernodes are responsible for inter-domain communication. At the inter-domain communication level, Bloom filters [4] are used to exchange information related to the domains in a light-weight manner. In this work, we extend our two-layered L-SCN architecture with additional mechanisms for intra-domain information propagation. These new mechanisms offer efficient information propagation in the intra-domain network. Please refer to [3] for more detail about L-SCN.

The previously specified intra-domain information propagation mechanism uses a consumer-driven technique to gather service provider related information such as available services and resources. This paper introduces two new mechanisms for intra-domain service and resource information propagation: provider-driven and event-driven service provider information propagation. In the provider-driven approach, the service provider nodes periodically broadcast information about available services and resources towards the intra-domain network. This enables the nodes in the domain to get domain-specific knowledge about existing services and available resources. This approach is different from the previously specified consumer-driven approach [3], because the previous mechanism needed to periodically request service and resources availability information from the service provider through Interests. The event-driven approach does not rely on any periodic broadcasting, but on sending service and resource availability information by a service provider as a reaction to a certain situation, which can be either a change in the provided services or a variation of available resources. This mechanism does not frequently propagate information to the network and, therefore, introduces low protocol overhead.

The paper is structured in the following manner. Section II presents important components of the NDN architecture and the related work. Section III describes the proposed routing mechanisms in a detailed manner. The evaluation results are illustrated in Section IV. Finally, we conclude in Section V.

II. Preliminaries and Related Work

NDN [5] introduces an evolutionary shift from the current host-centric Internet architecture towards a content-centric one. The goal is to bring solutions to problems arising from the significant development of services on the Internet in the last decades. From the interface (i.e., face) perspective, NDN has two message types, i.e., Interest and Data. The requester sends Interest messages for content, which contain a unique content identifier identifying the requested content. Data messages are a reply to the Interest messages, and they carry the requested content. NDN nodes use three main components to handle the Interest and Data forwarding process, namely the Forwarding Information Base (FIB), Pending Interest Table (PIT), and Content Store (CS), which provide the basic functionality

of the NDN node. A FIB stores a match between outgoing interfaces and name prefixes. Using the information stored in the FIB, a node can forward an incoming request through an interface leading towards content providers responsible for a given prefix. A FIB has the same purpose as routing tables in the current Internet. However, the aggregation is performed on the content level in a similar way as routing tables match destination networks with respective outgoing interfaces. The PIT keeps track of further forwarded Interest messages that have not yet been satisfied by the corresponding Data message. A PIT allows nodes to forward incoming Data messages along the same faces that previously forwarded requests for a given content. Finally, the CS is a cache that stores incoming Data messages; it enables incoming Interest requests to be satisfied by locally cached objects.

Service-Centric Networking (SCN) [2] extends ICN with service support capabilities. Several works focus on centralized approaches, in which a centralized controller having a complete view of the network and manages forwarding and matches consumers with appropriate service providers according to an arbitrary objective function [6], [7]. The centralized approaches come with a significant drawback, which is the single point of failure. Our proposed mechanisms do not rely on a centralized approach.

Service over Content-Centric Routing (SoCCeR) [8] integrates ICN with Ant Colony Optimization (ACO) [9] materializing service forwarding based on the information delivered by the ACO-based mechanism. SoCCeR gathers information on a per-service basis by randomly selecting a service and requesting service status information. SoCCeR periodically requests selected services to gather information on service providers. All nodes, through which the special request-reply messages were forwarded, store the provider related information for future forwarding decisions. Therefore, such a solution is only suitable for a network with a low number of available services, because it requests information about each service individually, and, hence, suffers from high overhead. Our proposed solutions are suitable for a large number of services. Other ACO-based ICN routing mechanisms [10]–[12] bring additional features such as energy-efficiency or QoS. However, they inherit the disadvantages of ACO-based solutions.

CCNxServ [13], which is based on CCNx [14], extends ICN by introducing service support. CCNxServ integrates a network service component called NetServ into CCNx. It allows for the dynamic deployment of services in the network. The CCNxServ service provider has to request a given service executable from the network and deploy it prior to the actual execution. CCNxServ has design-related performance issues caused by fetching and deploying the service before execution. Moreover, NetServ, being the core component of CCNxServ, is host-centric, which does not comply with the ICN architecture. Our proposed mechanisms entirely rely on ICN primitives.

Serval [15] introduces a new service layer between the transport and the network layer of the TCP/IP protocol stack. The introduced layer enables applications to communicate by using service names. To enable forwarding decisions, Serval builds a dedicated service table to map service names to the corresponding network addresses. In Serval, special routers are responsible for locating the best available service replica, these centralized routers introduce a single point of failure. Moreover, Serval fundamentally changes the TCP/IP stack, which makes the integration of Serval into the current Internet architecture difficult.

III. Communication Mechanisms

Currently, the L-SCN architecture uses a consumer-driven approach to propagate service provider related information in the network. The consumer-driven mechanism propagates information based on the request-response principle. The previous mechanism is presented in [3] and was briefly described in Section I. This Section presents the two newly designed routing mechanisms for provider information propagation: provider-driven and event-driven. In the provider-driven mechanism, a service provider periodically propagates service and resource availability information in the intra-domain network. The event-driven mechanism propagates service and resource availability information upon certain events, e.g., high resource utilization on a service provider. The provider-driven mechanism is more accurate, because it sends information with high granularity. The event-driven mechanism is typically slightly less accurate, while it sacrifices high information granularity to introduce lower overhead.

A. Provider-driven Mechanism for Service Provider Information Propagation

The first strategy for service provider information propagation relies on a push-based paradigm. The service providers periodically push their information into the intra-domain network. This is achieved through special Interest messages. They propagate information about available resources and services provided by a given service provider. Three types of Interest messages are used to materialize this protocol: Provider Availability Information (PAI), Service Availability Information (SAI), and Resource Availability Information (RAI). PAI messages are used by service providers to announce node availability information such as joining or leaving a given domain. SAI messages inform the intra-domain network about all available services in the network. RAI messages are sent periodically by the service providers to notify the intra-domain network nodes about available resources at a given moment of time for a given service provider. We envision that a communication protocol built upon these three message types is suitable for domains quickly instantiating or disposing services and having highly variable resource utilization.

Bloom Filter: SAI provides the intra-domain network with the currently available services. SAI messages are transmitted periodically to the entire domain. To achieve low overhead transmission for currently available services, the SAI message employs Bloom filters [4] to propagate the information on available services. Bloom filters are memory-efficient probabilistic data structures presenting a set of elements in a compact way. Bloom filters are widely used in ICN [16],

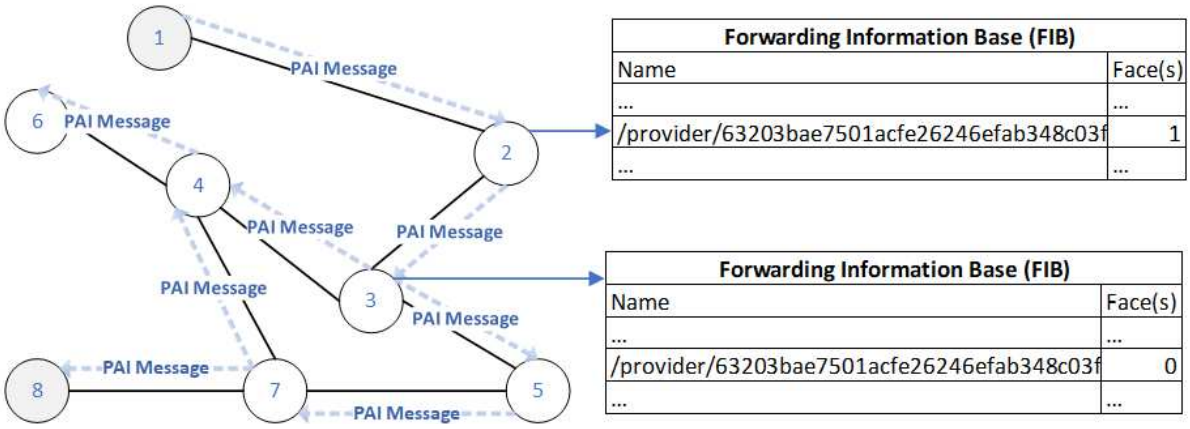

Fig. 1. PAI message propagation

[17]. Bloom filters support two operations, i.e., *insert* and *check*. Insert adds an element to the Bloom filter, while check indicates whether a given element is in a Bloom filter. Bloom filters use a bit vector to store a list of elements; prior to the storage, the elements are hashed with an arbitrary number of hash functions. The Bloom filter can be queried by checking whether a given element exists in the Bloom filter. Bloom filter queries do not produce false negatives (i.e., a Bloom filter cannot deny the existence of the previously inserted elements), but can produce false positives (i.e., a Bloom filter can confirm an element that was not inserted). Every Bloom filter comes with a particular trade-off among the size of the bit vector, the number of elements, the probability of false positives, and the number of deployed hash functions.

Protocol Messages: The following sections will present the three protocol messages: PAI, SAI, and RAI.

1) PAI: The service provider uses the PAI Interest messages to communicate important events such as joining and leaving the network. Again, the mechanism is push-based. Therefore, no reply messages to the Interest messages are sent. The PAI message uses a specific naming convention. It is composed of four parts. It starts with two keywords “bcast” and “pai”. They indicate that a given message is of the broadcast type (i.e., bcast) and carries the PAI signaling. The first two components are followed by the service provider’s unique identifier. The message name concludes with the desired signaling message. There are two key PAI signaling messages: join and quit. A join signaling message is sent by a service provider to announce (i.e., within a domain) that the service provider joins this domain. The quit signaling message is sent by a service provider to indicate that the service provider quits the domain. The PAI message structure is given by the following naming convention: “/bcast/pai/[providerID]/[join or quit]”. In particular, a join signaling message looks as follows: /bcast/pai/63203bae7501acfe26246efab348c03f/join, where 63203bae7501acfe26246efab348c03f is the identifier of a service provider. Upon forwarding of the join PAI, the intermediate nodes populate their FIBs acknowledging the presence of a new node in the system (e.g., 63203bae7501acfe26246efab348c03f) and configuring faces leading towards this destination (i.e., using the PAI face of arrival). In case of a quit signaling message, the intermediate nodes remove all entries corresponding to the given provider from their FIBs. Let us consider the following example. Fig. 1 illustrates a domain composed of 8 nodes. Node 1 (in grey) is a service provider joining the network and originating a PAI join message. As shown in Fig. 1, the PAI message is propagated through the entire domain. Therefore, the FIBs of all the nodes in the domain are populated by entries indicating the face, through which the newly joining service provider can be reached. Furthermore, Fig. 1 displays the FIBs of nodes 2 and 3. The inserted FIB entry begins with the keyword ”provider” followed by the provider identifier gathered from the PAI message. In the case of a quit signaling message, the nodes remove all their FIB and PIT entries corresponding to the quitting service provider. The PAI messages enable proper joining and leaving operations of service providers. The provider identifier is used to distinguish messages sent by different service providers. The forwarding of service requests is based on the service identifier.

2) SAI: SAI Interest messages are periodically broadcast by the service providers. They carry a Bloom filter containing the currently available services offered by a given node. The SAI Interest name follows a specific naming convention to indicate that the given Interest is of the SAI type. The SAI message starts with the prefix ”bcast” followed by the keyword ”sai”. The prefix ”bcast” again indicates that this is a broadcast Interest message, while the ”sai” keyword specifies the SAI request type. The prefix and keyword are followed by the identifier, which indicates the originating service provider. The provider identifier is a randomly generated identifier that uniquely identifies a given service provider in the intra-domain network. The Interest message also holds a timeout, which indicates the validity period of a given Interest message. As regular Interest messages, the SAI messages are stored in PIT tables of the intermediate nodes prior to forwarding. A node receiving a SAI message broadcasts it through all faces except the message face of arrival. Nodes receiving multiple messages

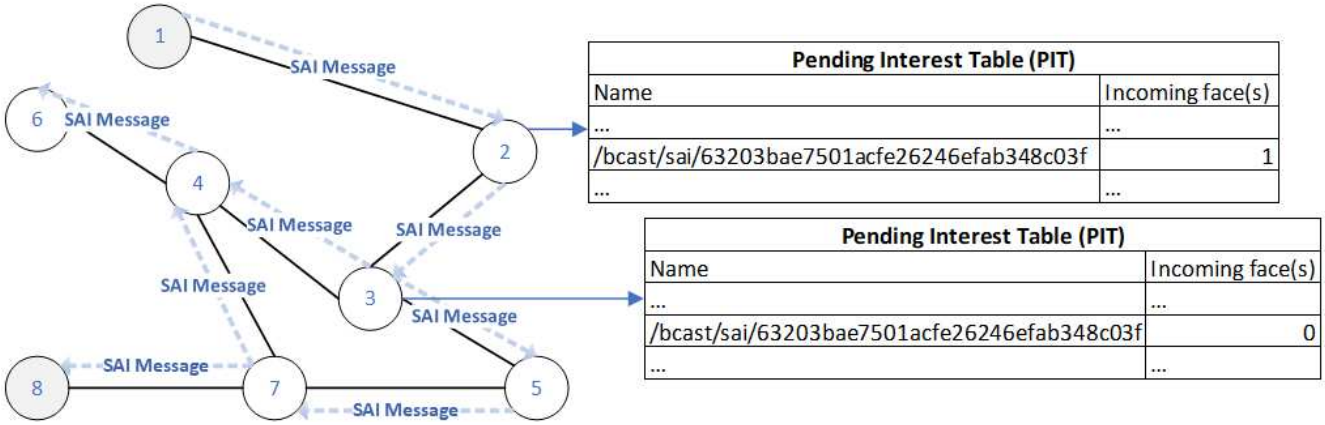

Fig. 2. SAI message propagation

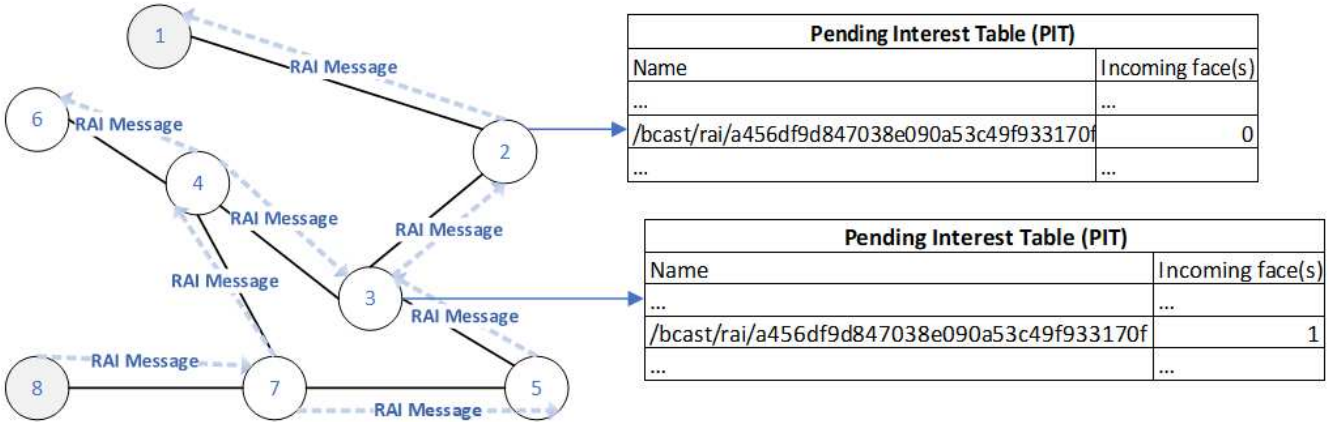

Fig. 3. RAI message propagation

from distinct service providers populate the PIT table by adding a new entry for every encountered service provider. Fig. 2 shows a domain with two service provider nodes 1 and 8 (grey circles). The service providers broadcast SAI messages to the entire domain. Fig. 2 shows the SAI broadcast of the service provider 1 and the resulting population of the FIBs of nodes 2 and 3. The SAI Interest message is propagated to the entire domain, while the carried nonce guarantees loop-freeness of the broadcast operation.

The PITs of two nodes (2 and 3) for our previous example are shown in Fig. 2. Upon forwarding of the SAI message, all nodes populate their PITs using the incoming Interest message. Nodes 2 and 3 (i.e., other nodes as well) use the entry to make a forwarding decision if the FIB does not contain forwarding information for a given service request. The PIT contains the Interest message with the corresponding outgoing face leading to every service provider. A SAI PIT entry is linked to a service provider by the unique identifier carried in the Interest name. In Fig. 2, the PIT of node 2 contains a new entry that has been added upon the forwarding of the SAI Interest. The entry follows the SAI naming convention, in which the unique identifier of a given node is carried as the last component. The second column of the PIT contains the incoming face(s) of the Interest message. Using this information, a forwarding node can link the Interest to its sender and also find out a face, over which a given sender is reachable.

3) RAI: RAI Interest messages contain resource availability information (e.g., CPU, GPU, RAM). Service providers periodically broadcast RAI messages within the entire domain. A RAI message contains the current state of resource availability on a given service provider. Similarly to SAI messages, RAI messages follow a defined naming convention. They start with the keywords ”bcast” and ”rai” followed by the service provider identifier. Fig. 3 illustrates the RAI propagation process originated by service provider 8. The RAI message is propagated through the entire domain. The intermediate forwarding nodes save the RAI message in the PIT tables and broadcast it further. The intermediate forwarding nodes broadcast a RAI message among all faces except the message face of arrival. In this example, the last component of the RAI message is a456df9d847038e090a53c49f933170f, which is the service provider identifier of node 8.

Service Interest Forwarding: Fig. 4 illustrates the forwarding process of an incoming service request Interest message. First, the nodes check whether there is an entry in the FIB for the requested service. If a FIB entry is available, the request

will be forwarded using the FIB entry. If no entry is available in the FIB, the node will query the stored Bloom filters to find out which service providers offer a given service. The service request is then forwarded to a service provider having the highest amount of available resources. A FIB entry for the corresponding service is added to the FIB. This enables further requests for the given service to be directly satisfied by the FIB. However, the entries in the FIBs are updated when new SAI and RAI messages arrive. Furthermore, the FIB entries that have not been used for a certain configurable amount of time are removed from the FIB. This enables keeping only relevant and recently requested services in the FIBs.

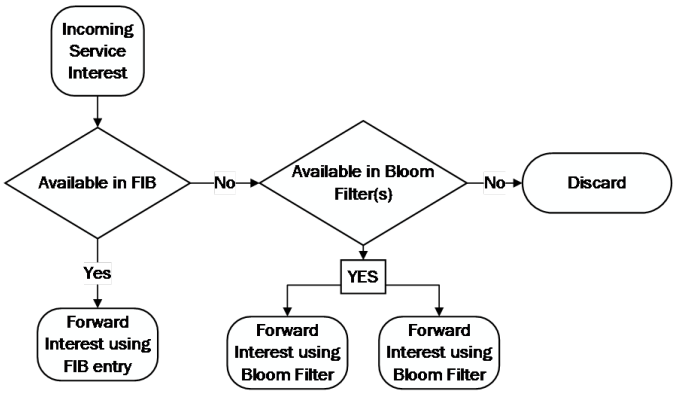

Fig. 4. Interest forwarding decision for the provider-driven mechanism

B. Event-driven Mechanism for Service Provider Information Propagation

The event-driven mechanism propagates service provider information about available services and resources based on an event-driven approach. Event-driven means that the propagation is initialized as a reaction to specific events that occur on service providers. In our context, an event is a change in the current status of the service provider. A service provider starts sending information about its available resources and services, if significant changes occur in the currently available services or available resources. The service provider propagates its available service information and resource status information in the case of changes with special signaling Interest messages. Two messages are used by the service provider for this purpose. They are called Service Availability Changes (SAC) and Resource Availability Changes (RAC), which indicate a change in service provider status in relation to available services and resources respectively. RAC and SAC are not sent periodically, but are based on the event-driven approach.

Protocol messages: The following sections present the SAC and RAC protocol messages.

1) SAC: Service Availability Changes (SAC) are Interest messages sent by a service provider to inform the network about updates on available services. The service providers have two possible data structures to store the list of available services in the SAC message. The service provider can send the list of available services provided as raw data or Bloom filters. The service provider node decides what data structures shall be used in the SAC message. The choice of the data structure mainly depends on the number of available services and the frequency of changes in offered services. When a service provider joins the network, it either sends the available services as raw data or Bloom filters. Further, the SAC message is not broadcast periodically, but its transmission is based on an event-driven approach.

There are two events that may occur with respect to service creation, namely *add* and *remove* indicating the creation or disposal of a service respectively. Upon a service creation event, the service provider either broadcasts a SAC message requesting the network to recognize the newly created service or a Bloom filter containing the set of offered services on a given service provider. In case of a service disposal event, the service provider either broadcasts a SAC message indicating the disposal of a given service or a Bloom filter with the new reduced set of offered services.

Message handling of the service *add* and *remove* events is similar. Upon an add event, the SAC message will be immediately sent to the network. The immediate broadcast of the message allows the entire domain to quickly recognize a new service on a given service provider. Upon the service removal event, the SAC message is only sent when the disposed service was recently solicited. Otherwise, the message will be sent upon receiving a request for the service. The SAC Interest message adopts a specific naming convention composed of five parts. The SAC Interest name starts with the keywords ”bcast” and ”sac” (/bcast/sac/) indicating that this is a broadcast message carrying a SAC Interest. The next component indicates the nature of the SAC signaling message, which can be of type ”add”, ”remove”, or ”bloomfilter”. The ”add” keyword indicates that the Interest handles the service creation, the ”remove” keyword specifies that the Interest contains disposed service identifiers, while the ”bloomfilter” keyword indicates that the Interest contains a new Bloom filter with the set of currently available services on a given service provider. The fourth component contains the provider identifier. Finally, the fifth component is provided or neglected depending on the nature of the SAC message. It is neglected in Bloom filter SAC messages; in this case, the Interest message, looks as follows: /bcast/sac/[providerID]/bloomfilter/, where [providerID] is the service provider identifier. In the case of add or remove SAC signaling messages, the last part contains the list of available services as name components. For example, for an add message the Interest name is structured as follows: /bcast/sac/ [providerID] /add/getweatheravg/getweatherinfahrenheit/getweather. It contains ”bcast” and ”sac” to indicate a SAC broadcast message. The third component is the identifier of the service provider node followed by the keyword ”add” indicating that this is a SAC message of type ”add”. The following components (getweatheravg, getweatherinfahrenheit, getweather) form the set of available service identifiers. The corresponding remove SAC message has a similar structure, but ”remove” replaces

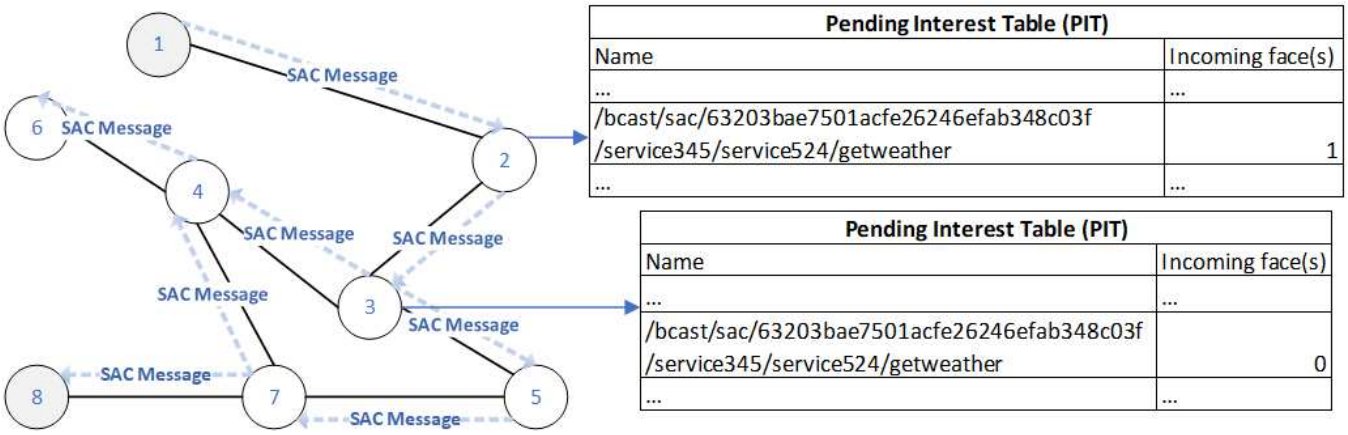

Fig. 5. SAC message propagation

the ”add” keyword. The SAC messages sent by the service providers are propagated across the entire domain. Before forwarding a SAC Interest message, the intermediate nodes store the Interest messages. Upon an add or remove SAC message, the respective service identifiers transported with the Interest message are added or removed from the existing table entries. If no entry exists for a given service provider, a new entry will be created. Fig. 5 displays the SAC propagation of service provider 1. Node 1 broadcasts the SAC add message containing three services. The Interest message is propagated through the whole domain. Before forwarding the SAC message, intermediate nodes store or extend the PIT entry corresponding to a given service provider. The entry contains the identifier of the SAC message and the service provider identifier followed by the currently available service identifiers. The SAC message incoming face indicates that such a face can be used to access a given service provider.

2) RAC: The Resource Availability Changes (RAC) Interest is a signaling message communicating that a given service provider is overloaded for a certain time period. The RAC message is composed of four parts. It starts with two component keywords ”bcast” and ”rac” indicating a RAC Interest broadcast message. The Interest name is followed by the service provider identifier and the RAC message type. The RAC message can be of two types: busy or available. The busy RAC message indicates that a given service provider is currently not available for some period of time due to the exhaustion of resources. The subsequent service requests should, therefore, not be forwarded towards this service provider. The available RAC message indicates that the service provider is again accessible. Therefore, subsequent service requests may be again forwarded to this service provider. The RAC busy messages are stored in the PIT and will remain there until the RAC timeout has expired or a RAC available message from a given service provider has arrived. In such a case, the busy RAC entry is removed from the PIT. Consequently, the service provider is no longer considered busy in terms of resource availability.

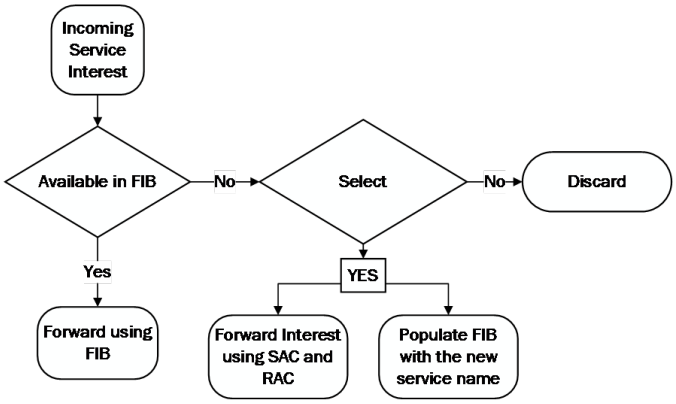

Fig. 6. Interest forwarding decision for the event-driven mechanism

Service Interest Forwarding: Fig. 6 illustrates the forwarding procedure of an incoming service Interest request. First, the intermediate nodes check whether there is an entry in their FIB for the corresponding service. If the FIB entry exists, the service request is regularly forwarded using the FIB. Otherwise, if no FIB entry exists, the SAC and RAC messages stored in the PIT are used to find the best service provider for a given service. Furthermore, a new entry for the given service is added to the FIB. The FIB entry permits subsequent requests corresponding to a given service to be directly forwarded using the FIB without employing the knowledge gathered from RAC and SAC messages. The FIB entries are also updated upon receiving RAC and SAC messages. Moreover, FIB entries not solicited for an extended period of time are disposed.

IV. Evaluation and Results

This section presents the evaluation results of the two newly introduced mechanisms (i.e., event-driven and provider-driven) compared to the existing consumer-driven solution [3]. We have conducted the evaluations in ndnSIM [18], which is an NDN simulation framework based on the NS-3 network simulator.

A. Evaluation Scenario

We have provided an evaluation network of 70 nodes. The network topology is illustrated in Fig. 7. It resembles the Internet topology at a small-scale having high degree connectivity nodes and low degree leaf nodes. We have randomly selected 3 leaf nodes as service consumers and 5 leaf nodes as service providers. Each service consumer sends 100 service requests with a frequency of one request per second. In total, 300 distinct service requests are sent by the service consumers that need to be satisfied by service providers. The processing time of the service request is uniformly distributed between 1000 and 1500 ms. During the simulation, we evaluate the service processing time, which is understood as the time elapsed between sending the service request and receiving the reply from the provider. The periodic broadcast of the consumer-driven and provider-driven approaches is set to 10 s. The busy threshold for the event-driven approach is set to 5 service requests waiting in the processing queue, and a busy message with a validity period of 5 s. These parameter values and this topology represent a realistic setup.

We have repeated the execution of this scenario ten times and compared the approaches in terms of processing time of the requests and generated overhead on a statistical basis.

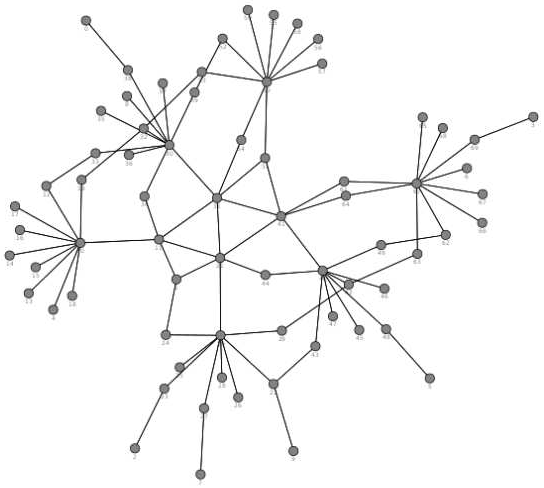

Fig. 7. Topology composed of 70 nodes

B. Evaluation Results

We have compared the newly presented provider-driven and event-driven mechanisms against the existing consumer-driven approach. Fig. 8 reveals the mean processing time of the three mechanisms with its respective confidence intervals for the mean value. Processing time is defined in the previous (i.e., IV.A) section. The bars show the mean processing time for the 300 requests and the horizontal lines mark the confidence intervals for the mean.

The confidence intervals for the consumer and provider-driven mechanisms are overlapping (Fig. 8, horizontal lines), suggesting insignificant differences between the mean processing time of these two strategies. The event-driven approach's confidence intervals are not overlapping with the confidence intervals (horizontal lines) of the two other approaches in Fig. 8. This suggests a significant difference between the mean processing time of the event-driven approach and the two other approaches. The consumer and provider-driven mechanisms have a mean processing time of 1427 ms (±20 ms) and 1417 ms (±19 ms) respectively. The results of these two strategies in terms of mean processing time are close together, because both mechanisms request or periodically push service provider resource availability information to the network. The event-driven approach does not use periodic resource availability information propagation, hence its mean processing time for the 300 requests is 2076 ms (±53 ms). The comparatively high mean processing time of the event-driven mechanism is because of its information propagation strategy, which propagates information only in case of service provider overload. The advantage of the event-driven approach is its low overhead (i.e., number of protocol messages), as shown in Fig. 9.

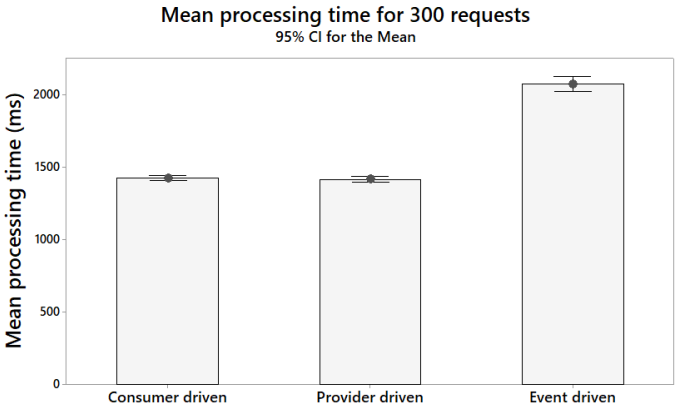

Fig. 8. Mean processing time for the 300 requests and the respective confidence intervals (CI) for the mean

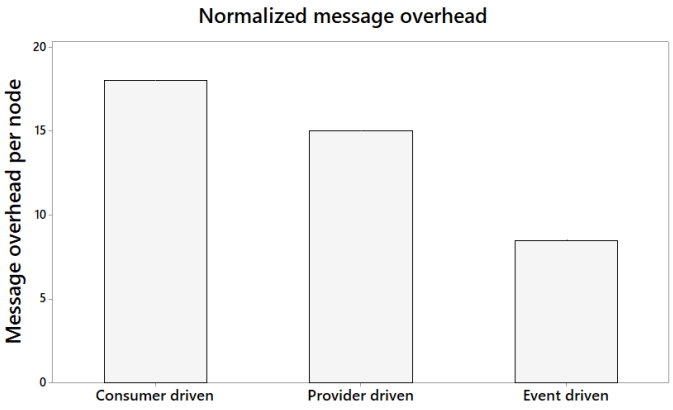

Fig. 9. Normalized message overhead

Fig. 9 illustrates the Normalized Message Overhead (NMO) of the three aforementioned approaches. We have normalized the overhead (i.e., number of protocol messages) to draw

a fair comparison between the mechanisms. The NMO is computed by $NMO = \frac{M}{N}$, where M is the number of protocol message transmissions and N the number of unique nodes that have received the message. NMO reflects the overhead cost per node. The respective NMO values for the three strategies are 18, 15, and 8.5. The event-driven approach has the lowest NMO value, because it does not perform periodic broadcasting, but only broadcasts in case of high overload. The consumer and provider-driven strategies have similar NMO values. The provider-driven strategy is better in terms of NMO compared to the consumer-driven approach. This is due to the request-response nature of the consumer-driven strategy, which generates more overhead.

Fig. 10 shows the Normalized Performance Ratio (NPR), which is an index combining the performance of the three mechanisms in terms of processing time and protocol message overhead. The higher the index value is, the better is the performance achieved by the aforementioned mechanisms. The index is computed as a product of the average processing time and the normalized message overhead divided by the resulting number of satisfied requests. The NPR is computed by the following formula: $NPR = \frac{NSSR}{MPT \cdot NMO} \cdot 100$, where NSSR is the number of satisfied service requests, MPT is the mean processing time, and NMO is the normalized message overhead. We multiply it by 100 to remove decimal points. Fig. 10 shows the NPR values, which are 1.17, 1.41, and 1.7 respectively for the three mechanisms. The event-driven approaches processing time value is the highest, however, taking into account the overhead in NPR, the event-driven mechanism delivers the best NPR index value.

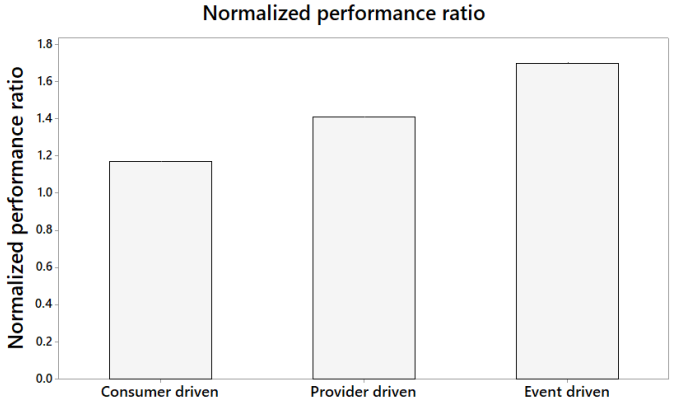

Fig. 10. Normalized performance ratio

V. Conclusions

In this work, we have presented new mechanisms for SCN, i.e., event-driven and provider-driven service provider information propagation. The event-driven approach propagates service provider related information to the network only upon an event (i.e., an overloaded service provider). In the provider-driven approach, the service provider is responsible for sending its status information to the network, which results in better mean processing time due to the fine-grained information propagation. We have compared our new approaches against the existing consumer-driven approach. The simulation results show that the new approaches offer better results compared to the existing ones in terms of mean processing time and protocol overhead. The provider-driven approach pushes information to the network, hence enabling faster propagation of service provider status information. The event-driven approach significantly lowers the protocol overhead and offers better trade-off ratio between overhead and mean processing time.

References

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, Jul. 2012.

[2] T. Braun, V. Hilt, M. Hofmann, I. Rimac, M. Steiner, and M. Varvello, "Service-Centric Networking," in *2011 IEEE International Conference on Communications Workshops (ICC)*. IEEE, jun 2011, pp. 1–6.

[3] M. Gasparyan, T. Braun, and E. Schiller, "L-SCN: Layered SCN architecture with supernodes and Bloom filters," in *2017 14th IEEE Annual Consumer Communications and Networking Conference, CCNC 2017*. IEEE, jan 2017, pp. 899–904.

[4] B. H. Bloom and B. H., "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, jul 1970.

[5] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[6] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," in *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE, 2009, pp. 124–131.

[7] R. Dutta, G. N. Rouskas, I. Baldine, A. Bragg, and D. Stevenson, "The SILO Architecture for Services Integration, controL, and Optimization for the Future Internet," in *2007 IEEE International Conference on Communications*. IEEE, jun 2007, pp. 1899–1904.

[8] S. Shanbhag, N. Schwan, I. Rimac, and M. Varvello, "SoCCeR: Services over content-centric routing," in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*. ACM, 2011, pp. 62–67.

[9] K.-L. Du and M. N. S. Swamy, "Ant Colony Optimization," in *Search and Optimization by Metaheuristics*. Bradford Company, 2016, pp. 191–199.

[10] C. Li, W. Liu, L. Wang, M. Li, and K. Okamura, "Energy-efficient quality of service aware forwarding scheme for Content-Centric Networking," *Journal of Network and Computer Applications*, vol. 58, pp. 241–254, dec 2015.

[11] Q. Huang and F. Luo, "Ant-colony optimization based QoS routing in named data networking," *Journal of Computational Methods in Sciences and Engineering*, vol. 16, no. 3, pp. 671–682, oct 2016.

[12] J. Lv, X. Wang, K. Ren, M. Huang, and K. Li, "ACO-inspired Information-Centric Networking routing mechanism," *Computer Networks*, vol. 126, pp. 200–217, oct 2017.

[13] S. Srinivasan, A. Singh, D. Batni, J. W. Lee, H. Schulzrinne, V. Hilt, and G. Kunzmann, "CCNxServ: Dynamic service scalability in information-centric networks," in *2012 IEEE International Conference on Communications (ICC)*. IEEE, jun 2012, pp. 2617–2622.

[14] V. Jacobson, D. K. Smetters, J. D. Thornton, M. Plass, N. Briggs, and R. Braynard, "Networking named content," *Communications of the ACM*, vol. 55, no. 1, p. 117, 2011.

[15] E. Nordström, D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Y. Ko, J. Rexford, and M. J. Freedman, "Serval: an end-host stack for service-centric networking," pp. 7–7, 2012.

[16] A. Marandi, T. Braun, K. Salamatian, and N. Thomos, "Pull-based Bloom Filter-based Routing for Information-Centric Networks," in *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, jan 2019, pp. 1–6.

[17] M. Gasparyan, A. Marandi, E. Schiller, and T. Braun, "Fault-Tolerant Session Support for Service-Centric Networking," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. Washington D.C.: IEEE, 2019.

[18] A. Afanasyev, S. Mastorakis, I. Moiseenko, and L. Zhang, "ndnSIM," 2017. [Online]. Available: https://ndnsim.net